04/12/2023
Monday

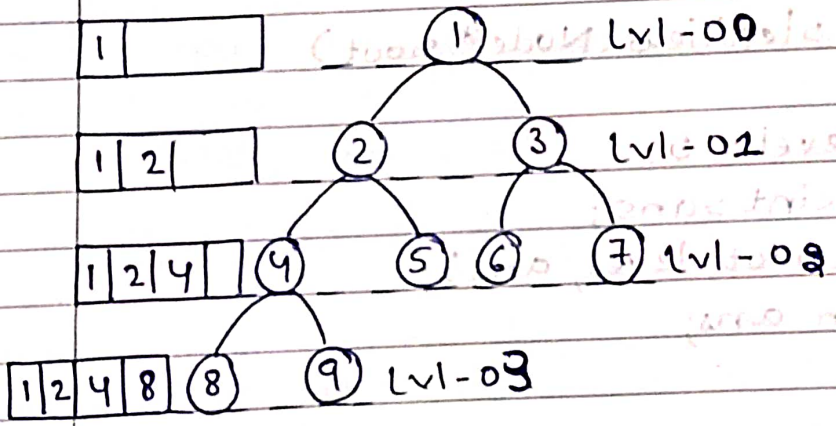
* **Left View of Binary Tree -**

Left view of binary tree is a set of nodes visible when tree is visited from left side.

- Har level ka first element comes under left view of binary tree.

I/P - 1 2 3 4 5 6 7 8 9

O/P - 1 2 4 8



Approach - let a vector v,

If v.size() equals current level of binary tree, then push element in vector v.

```
if (v.size() == level)
    v.push_back(root->data);
```

Code -

```
class Solution {
  public:
void solve (Node* root, int level, vector<int> &ans)
{
    if (root == NULL) return;

    if (level == ans.size())
       ans.push_back (root->data);

    solve (root->left, level+1, ans);
    solve (root->right, level+1, ans);
}

vector <int> leftView (Node * root)
{
    int level = 0;
    vector<int> ans;
    solve (root, level, ans);
    return ans;
  }
};
```

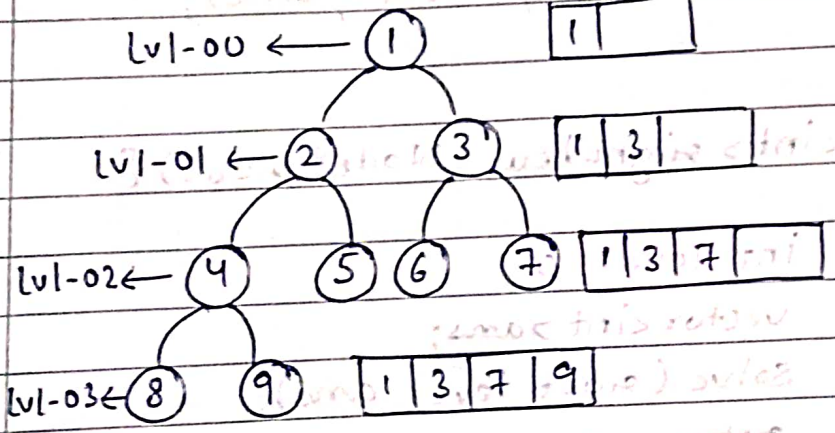
* Right view of Binary Tree -

Right view of binary tree is set of nodes visible when tree is viewed from right side.

- Har level ka last i.e. rightmost element comes under right view of binary tree.

I/P - 1 2 3 4 5 6 7 8 9

O/P - 1 3 7 9



Approach - If v.size() equals current level of binary tree, then push root->data in vector.

```
if (v.size() == level)
    v.push_back(root->data);
```

= Twist is, unlike left view, yha pr right subtree ki recursive call pehle lgegi, then left subtree ki because rightmost elements ya nodes are considered in right view.

Code-

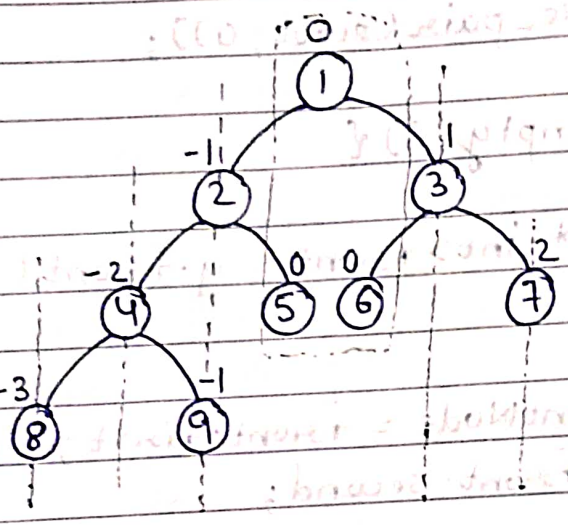
```
class Solution {
public:

void solve (Node* root, int level, vector<int> &ans)
{
    if (root == NULL) return;

    if (level == ans.size())
        ans.push_back (root -> data);

    solve (root -> right, level+1, ans);
    solve (root -> left, level+1, ans);
}

vector<int> rightView (Node* root) {

        int level = 0;
        vector<int> ans;
        solve (root, level, ans);
        return ans;
    }
};
```

* Top view of Binary Tree -

Top view of a binary tree is the set of nodes visible when tree is viewed from top.

I/P - 1 2 3 4 5 6 7 8 9

O/P - 8 4 2 1 3 7



Approach -

Yha pe hum vertical levels create kr lenge.

- Root node ko let krenge level → 0.
- Root node ke left mei jayenge to level - 1 krdenge.
- Root node ke right mei jayenge to level + 1 krdenge.

= Map create krenge jo us level ke corresponding jo node top view me aayegi, use store krega.

= Queue ka use krenge jisme queue ka har box node ka data or us node ke corresponding jo level hai use store krega. Initially q.push(make_pair(root, 0));

→ level ka track rkh sake. That's why queue mei node and uske corresponding level ka pair create kr diya.

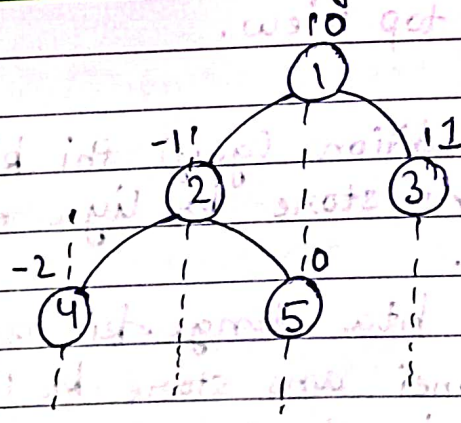
Code-

```
class Solution {
public:

void topview (Node* root, vector<int>&ans) {

    map<int, int> hdtoNodeMap;
    queue < pair<Node*, int> > q;

    q.push (make_pair (root, 0));

    while (!q.empty()) {

        pair<Node*, int> front = q.front();
        q.pop();

        Node * frontNode = front.first;
        int hd = front.second;

// if there is no entry for current vertical
// level or hd, then create an entry in map.
        if (hdtoNodeMap.find(hd) == hdtoNodeMap.end()) {
            hdtoNodeMap[hd] = frontNode->data;
        }

// current node ke child push krdo if exists.

if (frontNode->left != NULL)
    q.push(make_pair (frontNode->left, hd-1));

if (frontNode->right != NULL)
    q.push (make_pair (frontNode->right, hd+1));
}
```

```
for (auto i : hdtoNodeMap) {
        ans.push_back (i.second);
    }
}

vector<int> topView (Node* root)
{
    vector<int> ans;
    topview (root, ans);
    return ans;
}
};
```

→ Initially, root node pe hd i.e. vertical level 0 hai. (hd = 0)
rootNode → left = (hd - 1)
rootNode → right = (hd + 1)

→ let for binary tree -



Map

-2	4
-1	2
0	1
1	3

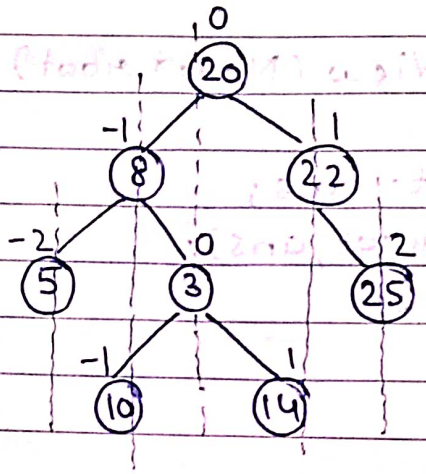
Queue - 1|0 2|-1 3|1 4|-2 5|0

→ level order traversal ka hi use kiya hai top view nikalne me, bs levels vertical hai.

* **Bottom view of Binary Tree -**

A node is included in bottom view if it can be seen when we look at the tree from bottom.

I/P -



O/P - 5 10 3 14 25

Approach -

Same as finding top view.

= Yha pr bs jo ek condition lgaiji thi ki agr ek level ke liye ans store kr liya map to fir se nhi krna.
Bs us condition ko hta denge ki agr ek level ke liye map mei ans store kr liya to use firse overwrite kr do.
Ese krne se map mei at the end vhi elements honge jo bottom view mei aa rhe hai.

Code -

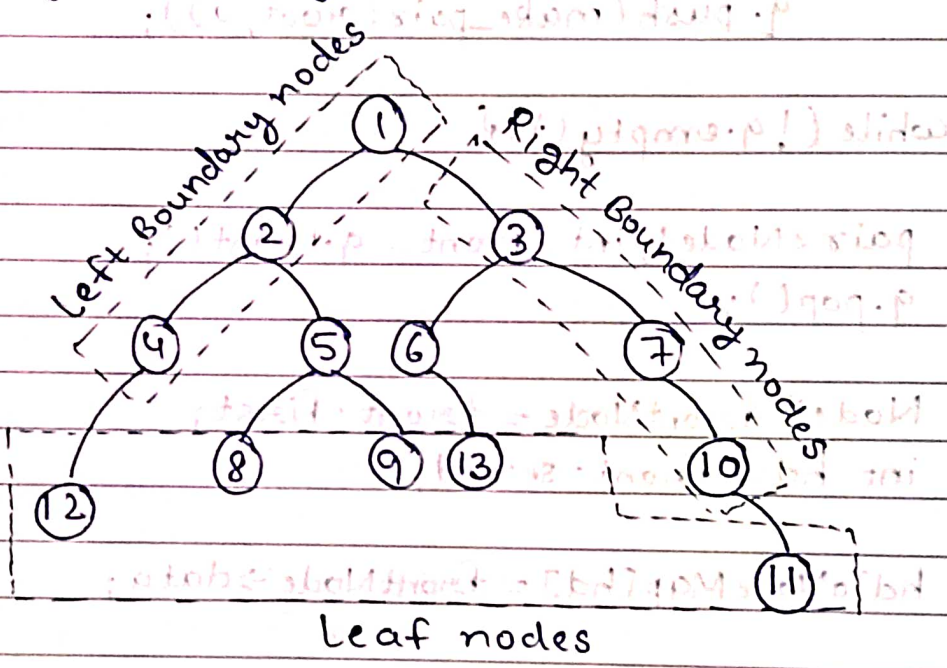
```
class Solution{
public:

void solve(Node* root, vector<int> &ans){

    map<int, int> hdToNodeMap;
    queue<pair<Node*, int>> q;

    q.push(make_pair(root, 0));

while(!q.empty()){

  pair<Node*, int> front = q.front();
  q.pop();

  Node* frontNode = front.first;
  int hd = front.second;

  hdToNodeMap[hd] = frontNode->data;

  if(frontNode->left != NULL)
     q.push(make_pair(frontNode->left, hd-1));
  if(frontNode->right != NULL)
     q.push(make_pair(frontNode->right, hd+1));
  }
for(auto i : hdToNodeMap){
     ans.push_back(i.second);
  }}
vector<int> bottomView(Node* root){
     vector<int> ans;
     solve(root, ans);
     return ans;
    }
};
```

* **Boundary Traversal of Binary Tree –**

Boundary of binary tree comprises –

- Left Boundary nodes
- Leaf nodes
- Right Boundary nodes

I/P –



O/P – 1 2 4 12 8 9 13 11 10 7 3

Approach –

- **Left boundary nodes** consists of leftmost nodes of binary tree. Agr leaf node mile to vha se return kr jao.

Important thing is agr left mei jana possible hai to left mei jao nhi to right mei jao but ek hi side ja skte hai either left or right.

- leaf nodes mei simply vo nodes aa jayegi jinka left or right dono NULL hai.

- Right Boundary nodes consists of rightmost nodes of binary tree. Agr leaf node mile to vha se return kr jao.
Important thing is agr right mei jana possible hai to right mei jao nhi to left mei jao. Agr right ki call lg gyi to left ki call nhi lgegi.

Code -

```
class Solution{
public:

void leftBoundary(Node* root, vector<int> &ans)
{
    if(root==NULL) return;

if(root->left==NULL && root->right==NULL) return;

    ans.push_back(root->data);
  if(root->left!=NULL) leftBoundary(root->left, ans);
  else leftBoundary(root->right, ans);
}

void leafBoundary(Node* root, vector<int> &ans){
    if(root==NULL) return;

if(root->left==NULL && root->right==NULL){
 ans.push_back(root->data);
}
leafBoundary(root->left, ans);
leafBoundary(root->right, ans);
}
```

```
void rightBoundary(Node* root, vector<int>&ans)
{
    if (root == NULL) return;

if(root->left==NULL && root->right==NULL) return;

if (root->right != NULL) rightBoundary(root->right, ans);
else rightBoundary (root->left, ans);
ans.push_back(root->data);
}

void boundaryTraversal (Node* root, vector<int>&ans)
{
    if(root==NULL) return;

    ans.push_back (root->data);
    leftBoundary (root->left, ans);
    leafBoundary(root->right,
    leafBoundary(root->left, ans);
    leafBoundary (root->right, ans);
    rightBoundary (root->right, ans);
}

vector<int> boundary(Node*root)
{
    vector<int>ans;
    boundaryTraversal (root, ans);
    return ans;
}
```

Answer ko 4 parts mei store krva liya -

(1) Root of binary tree.

(2) left Boundary wale function ki call to root->left.

(3) leaf Boundary wale function ki call to root->left and root->right. left and right ke liye alg alg isliye lgayi because single element ke case mei do baari print ya store ho jayega.
Ek baar Jb root store krvaya and dusri baar Jb leaf boundary wale function ki call lgi.
To prevent this, left and right subtree ki call alg alg lgayi.

(4) Right Boundary wale function ki call to root->right